Analysing the similarity of students' programming assignments

Tatjana Stojanovic

Department of software engineering,
Faculty of organizational sciences
University of Belgrade
Belgrade, Serbia
tatjana.stojanovic@fon.bg.ac.rs,
0000-0001-7191-6444

Sasa D. Lazarevic

Department of software engineering,
Faculty of organizational sciences
University of Belgrade
Belgrade, Serbia
sasa.lazarevic@fon.bg.ac.rs

Abstract— Having many assignments teachers tend to use automated grading systems to make the grading process more efficient. One of the problems while using such systems is detecting plagiarism. Many tools for detecting plagiarism have been developed and are still being improved. These tools can detect the percentage of program similarity, but cannot determine if the similarity is a product of plagiarism or not. In this paper, it is shown that high similarities between implementations are not always a product of plagiarism. Functions which may have a high similarity percentage among students are those for which solution was provided during the course, coding style used during the course, and small functions with a small number of possible solutions. For these functions, it is not possible to determine whether the code was plagiarized only by detecting code similarity.

Keywords— plagiarism, plagiarism detection, code similarity, automated grading systems

I. INTRODUCTION

The outbreak of the COVID-19 pandemic forced many faculties to change their courses in order to better adapt to the new circumstances. This pushed e-learning to become the dominant mean of teaching in a situation where physical contact is meant to be restricted. In recent years, many faculties had to adapt their courses to be online and the popularity of online courses is rapidly growing. The sudden increase in the volume of usage of e-learning platforms also brought new problems which teaching staff had to consider when conducting the faculty curriculums. One of the mentioned problems refers to the large number of exams which had to be reviewed and graded. In order to assess many students' programming assignments, teachers tend to use automated grading systems. One of the main problems with automated assessment is detecting plagiarism among the students' assignments.

Oxford University defines plagiarism as "presenting someone else's work or ideas as your own, with or without their consent, by incorporating it into your work without full acknowledgement".[4] It is important for the issue considered in this paper to emphasize that plagiarism refers not only to physical materials, but also to those in electronic form, as well as the fact that plagiarism can be intentional, but also unintentional. In the context of programming assignments, students are prone to take others' code and incorporate it into their assignments, often without changing even a single line and without fully understanding it. This can be especially problematic in exams which are meant to test students' problem-solving skills and their programming proficiency.

Tools for automated grading of programming assignments usually impose certain restrictions on the possible implementation of the students' solutions. These rules are most often concerning function prototypes, class names etc. Regarding strict rules when taking a test, several issues can be observed. Students must follow pre-determined blueprints while implementing the functions, which leaves them with little space to use their problem-solving skills. Types of automated tools which use static analysis usually expect a student to implement one of the model solutions and can be too restrictive [3], while tools which are test-based can be too strict while grading because they is unable to grade code which has any error.[6] It is important to estimate how strict the restrictions imposed on students should be, in order to be able to detect plagiarism correctly.

To detect plagiarism, many tools for detecting the similarity of programs have been developed. Some of the popular tools are MOSS, JPlag, Plaggie and others.[2][5][1] These tools compare all submissions against each other and usually provide a report containing the most similar programs. As stated, these tools only provide information about the similarity of programs, without determining why the programs are similar.

Here will be analyzed how the similarity of programs depends on the given assignments. Some functions are common in programming or there is a limited number of ways to implement them. Also, in the course, students are taught to implement a function usually in one manner. When students are required to implement such functions in an exam, it is likely for programs to be more similar. These functions will be referred to as common functions. When students are required to use their problem-solving skills and solve more complex problems which may differ considerably from those lectured during the course, it is expected that students will have more different solutions. These functions, which can have multiple implementations, will be referred to as non-common functions. In this work similarity of programs containing common functions as well as those containing non-common functions will be analyzed, in order to ascertain how assignments affect the usage of these tools and to prevent misusage of plagiarism tools. By knowing to what extent this alters results, it will be possible to determine which similarity percentage is abnormal and probably a result of plagiarism.

II. METHODOLOGY

Analysis of students' programming assignments was performed with assignments from an introductory programming course in C. Assignments are consisted of several functions to be implemented. Due to usage of automated grading tool, students were given prototypes of these functions and type definitions for structures. Some functions (i. e. printing array, matrix, finding a member in an array or a list...) are as same or similar to those shown during the course, while other functions are not provided in the course materials.

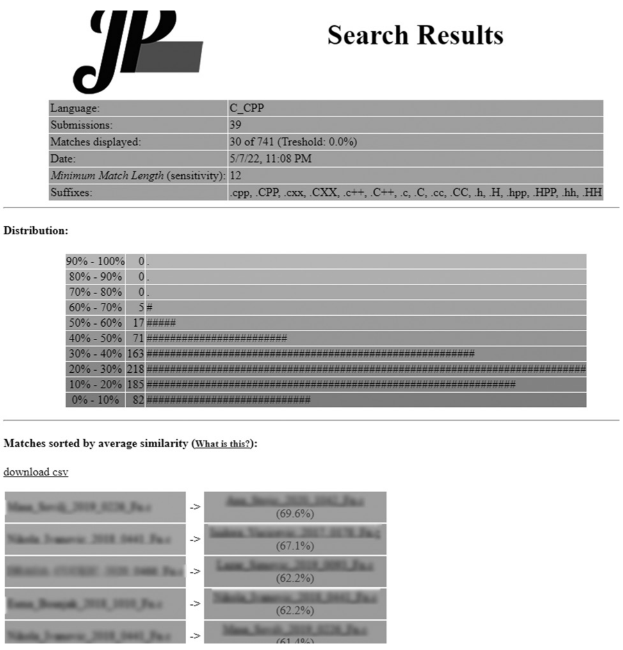
The similarity of assignments are analysed in four ways:

1. whole programs containing all functions,
2. only functions which are considered common or simple functions for which similarity is predicted to be high,
3. only functions which are considered non-common, functions which haven't been explained during the course and can have multiple solutions, thus lower percentage of similarity is predicted,
4. each function separately, which allows determining what kind of function had higher similarity and which had a lower percentage of similarity.

All analysed assignments were implemented by students at one exam. The exam was conducted at the faculty, while students were monitored by teachers. Nevertheless, the problem analyzed here can also be applied to situations where exams are conducted fully online. The assignment consisted of seven functions which students had to implement. Two of seven functions are considered common (i.e. print an array, find a node in the list etc.). Three of them are considered non-common, meaning that solutions to these or similar functions were not provided in the materials for the course. While two remaining functions cannot be grouped in either of the two.

JPlag tool was used for determining program similarity analysis because the report shows a number of matches grouped by the percentage of similarity, which allows for determining the average percentage. In Fig. 1, a screenshot of JPlag's report is shown. When comparing all submissions with this tool, sensitivity must be determined. Sensitivity represents the minimum number of tokens required to be counted as a matching section, and smaller values might lead to more false positives.[4] The sensitivity value cannot exceed the maximum number of tokens. Sensitivity for the whole assignments and files with more functions was set to the default value of 12 tokens. For files containing only one function sensitivity value of 4 tokens was used, because some functions were short and sensitivity above 4 token did not successfully determine similar parts of code.

In order to assess the similarity of these programs, for each student program were generated four types of files – a file containing all functions, a file containing only functions which are considered common or simple, a file containing only those which are non-common and a separate file for each function.



Search Results

Language:	C_CPP
Submissions:	39
Matches displayed:	30 of 741 (Treshold: 0.0%)
Date:	5/7/22, 11:08 PM
Minimum Match Length (sensitivity):	12
Suffixes:	.cpp, .CPP, .cxx, .CXX, .c++, .C++, .c, .C, .cc, .CC, .h, .H, .hpp, .HPP, .hh, .HH

Distribution:

90% - 100%	0	.
80% - 90%	0	.
70% - 80%	0	.
60% - 70%	5	#
50% - 60%	17	#####
40% - 50%	71	########################
30% - 40%	163	##
20% - 30%	218	###
10% - 20%	185	##
0% - 10%	82	############################

Matches sorted by average similarity (What is this?):

download csv

[redacted]	->	[redacted] (69.6%)
[redacted]	->	[redacted] (67.1%)
[redacted]	->	[redacted] (62.2%)
[redacted]	->	[redacted] (62.2%)
[redacted]	->	[redacted] (61.4%)

Fig. 1. JPlag report

III. RESULTS

Before the analysis, students' assignments were separated into the files as stated above. The total number of performed analyses is ten:

- 1 for the whole assignment containing 7 functions which students were asked to implement. Other auxiliary functions which students may have implemented were excluded,
- 1 for functions which are expected to be similar (common),
- 1 for functions which were not expected to be similar (non-common),
- 7, one for each function.

At Fig. 2 the vertical axis shows a percentage of total matches done, while the horizontal axis represents a group of similarity. For example, almost 40% of matches done with files which contained only common functions, have between 40 and 50% of similarity. In Table I, number of comparisons in each similarity group is shown, but results were separated in five groups instead of default ten. As it can be seen from Fig. 2 and Table 1, the similarity tends to be over 50%. For non-common functions, the similarity tends to be lower than 20%. Overall similarity tends to be lower than 30%. Meaning if the assignment consisted only of functions for which solutions are provided in materials or during the course, a higher similarity percentage can be expected. This similarity, however, is not necessarily the consequence of plagiarism since the common functions can be implemented in a limited number of ways (as explained before), so it is not possible to determine whether the code was plagiarised or not at all. This may affect the ability to use tools for determining plagiarism. On the other hand, assignments which require better understanding and problem-solving skills and for which many different solutions can be constructed make preventing and detecting plagiarism much easier, but also can lead to a higher number of failed assignments, since the non-common functions tend to be more difficult for students. It is also observed that a lower number of students have implemented these functions.

Similarity for whole assignments shows that for this particular exam plagiarised assignments can easily be observed, with a high percentage of similarity. But if the assignment contained only common functions, plagiarised assignments would not be easily detected.

TABLE I. ASSIGMENTS SIMILARITY REPORT

Similarity %	**Assignments comparisons**					
	Expected to be similar		**Expected not to be similar**		**Whole assignments**	
	Number	**Percentage**	**Number**	**Percentage**	**Number**	**Percentage**
0%–20%	70	9%	504	72%	267	36%
20%–40%	73	9%	178	25%	381	51%
40%–60%	357	46%	20	3%	88	12%
60%–80%	116	15%	1	0%	5	1%
80%–100%	70	9%	0	0%	0	0%

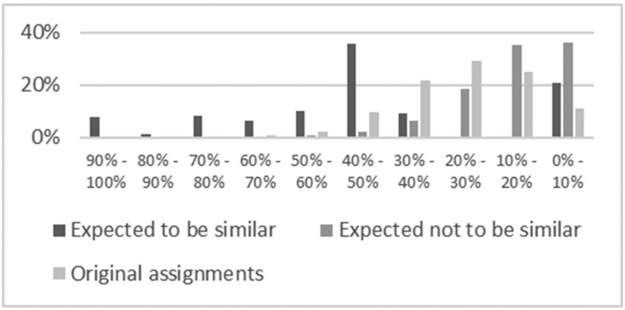


Fig. 2. Similarity report between files with all functions, files only with functions which are expected to be similar and between files only with functions which are not expected to be similar

In Fig. 3 and in Table II, the percentage for two functions which are expected to be similar is shown. The percentage of comparisons for each similarity level is shown. Functions names are:

- F1 and F2 – common functions (e. g. printMatrix, addNewNode…),
- F4, F5, F6 – non-common functions, functions previously unfamiliar to students (e. g. startGame, addMove, transformList…)
- F3, F7 – cannot be grouped in one of the previous two groups.

As it can be seen, the average percentage tends to be higher among common functions. For example, almost 90% of matches had a 90% of similarity of function for printing matrix (F1). For function F2, over 75% of total matches have similarities over 60%. For functions F4, F5, and F6, it can be observed that none of the functions has a high percentage of similarity matches (over 60%). Some of the functions like F5 and F4 have over 78% and over 40% of matches below 40% of similarity, where half of the comparisons for function F5 have similarity below 20%.

TABLE II. ASSIGMENTS SIMILARITY REPORT FOR EACH FUNCTION (%)

Similarity %	**Comparisons of functions (%)**						
	F1	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**
0%–20%	0%	4%	9%	15%	50%	4%	23%
20%–40%	0%	10%	27%	38%	29%	17%	12%
40%–60%	1%	8%	26%	26%	16%	31%	26%
60%–80%	0%	30%	21%	19%	5%	30%	20%
80%–100%	99%	48%	17%	3%	0%	6%	18%

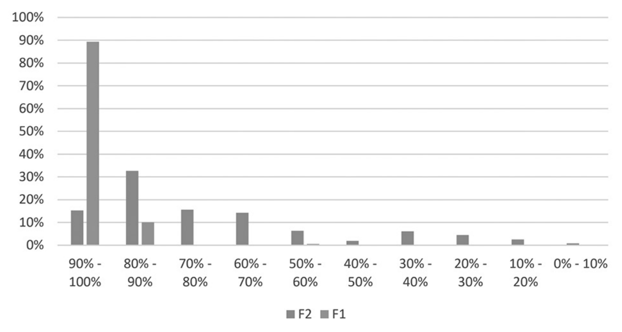


Fig. 3. Similarity between common functions

In Fig. 4, the number of percentages for three of the non-common functions is shown. As it can be seen, one of the functions is not similar at all to other implementations in almost 40% of the comparisons.

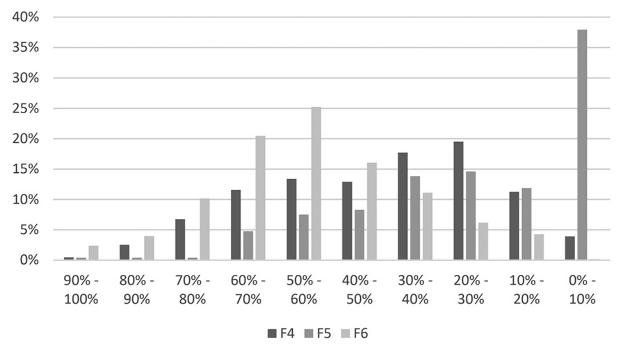


Fig. 4. Similarity between non-common functions

IV. DISCUSSION

When given functions for which solutions were given in materials or explained during the course, or similar, a higher percentage of similarity is expected. Also, for functions which have a limited number of possible implementations, we can expect the similarity to be higher. In this case, the high percentage of similarity does not necessarily mean that there had been plagiarism among the students' works. Plagiarism is harder to detect in this case.

With functions for which can be given more than one solution and for which there is not a solution provided during the course, a lower percentage of similarity is expected. For these kinds of functions, it is easier to determine whether implementations were a result of plagiarism. In this case, a high percentage of similarity between students' works usually means that they had plagiarised.

When a number of submissions are compared it is obvious that those new functions, which are not common, have a lower number of submissions. Also, implementing common functions like printing or reading from standard input cannot be avoided, same as simple functions for which students may give similar solutions. Teachers should be aware that the percentage of similarity depends on the earlier students' experience of solving those functions. When functions are the same as those presented during the course, a higher similarity is observed and it is hard to determine if this similarity is the consequence of plagiarism or the student's familiarity with the problem being solved.

In the mentioned tools, there is a possibility of excluding such common parts of the programs from the report, but when the programs mainly consist of simple or already familiar functions, the usability of these tools is questionable.

When given an assignment containing several functions which are familiar and several which could be considered non-common, tools for detecting plagiarism are effective in determining the assignments which contain plagiarism to a large degree, but smaller functions or partially plagiarized code may remain undetected.

Some tools for plagiarism detection such as MOSS, have ways to avoid grading common functions or shared code as plagiarism. MOSS enables the declaration of base files which e. g. could be provided by teachers. In addition, MOSS enables declaring a maximum number of assignments before declaring a part of code as common. For number n, if a code appears in n or more assignments, that part of the code isn't considered as part of plagiarized code. One of the cons of using MOSS is that it is an online-only tool which generates HTML response which isn't considered safe in terms of students' privacy. For this research, MOSS doesn't provide an aggregated report, making it troublesome to assess a normal percentage of similarity and its distribution.

V. CONCLUSION AND FUTURE WORK

In this paper, it has been shown how a previous knowledge of possible implementations for functions can result in similarly implemented solutions among the students. But also, using common functions is often necessary while implementing programs, and can't be avoided. With using programs such as JPlag, assignments which contain plagiarized code should be obvious, depending mainly on assignment requests.

Plagiarism tools and tools for automated grading are being used for a long time and improved through the years. While automated grading brings many benefits to the process of grading, it also limits the number of possible solutions for the assignments, making it harder to determine whether a similar code is a result of plagiarism or a coincidence.

When designing the assignments, teachers should have in mind not only the functions for which solutions are shown during the course, but should also consider how the complexity of the function affects the same issue. The simpler the function, the higher similarity should be expected, because a limited number of solutions are possible, while students are not motivated to implement the most creative solution, but only a working solution. But if functions are too complex, automated test tools could be too strict in terms of grading, not being able to partially grade code.

REFERENCES

[1] Ahtiainen, A., Surakka, S., & Rahikainen, M. (2006). Plaggie. Proceedings of the 6th Baltic Sea Conference on Computing Education Research Koli Calling 2006 - Baltic Sea '06. https://doi.org/10.1145/1315803.1315831

[2] Alex Aiken. MOSS (Measure Of Software Similarity): A System for Detecting Software Similarity. (n.d.). Stanford. Retrieved May 12, 2022, from https://theory.stanford.edu/~aiken/moss/

[3] K. Ala-Mutka. A survey of automated assessment approaches for programming assignments. Computer Science Education, 15(2):83–102, 2005.

[4] Plagiarism | University of Oxford. (2022). Plagiarism. https://www.ox.ac.uk/students/academic/guidance/skills/plagiarism

[5] Prechelt, L., Malpohl, G., & Phlippsen, M. (2000, March). JPlag: Finding plagiarisms among a set of programs. https://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf

[6] Stojanović, T., & Lazarević, S. (2021). Automated grading assignments in programming – advantages, problems and effects on learning. E-Business Technologies Conference Proceedings, 1(1), 100–104. Retrieved from https://ebt.rs/journals/index.php/conf-proc/article/view/77